

Combining Clustering with Moving Sequential Patterns Mining: A Novel and Efficient Technique*

Shuai Ma, Shiwei Tang, Dongqing Yang, Tengjiao Wang, Jinqiang Han

Department of Computer Science, Peking University, Beijing 100871, China
National Laboratory on Machine Perception, Peking University, Beijing 100871, China

{mashuai, tjwang, jqhan}@db.pku.edu.cn
{tsw, dqyang}@pku.edu.cn

Abstract

Sequential pattern mining is a well-studied problem. In the context of mobile computing, a special sequential pattern, moving sequential pattern that reflects the moving behavior of mobile users attracted researchers' interests recently. Moving sequential patterns can be viewed as a special type of conventional sequential pattern with the extension of support. Mining moving sequential patterns has great significance for effective and efficient location management in wireless communication systems. In this paper a novel and efficient technique is proposed to mine moving sequential patterns. Firstly the idea of clustering is introduced to process the original moving histories into moving sequences as a preprocessing step. And then an efficient algorithm, called PrefixTree, is presented to mine the moving sequences. Performance study shows that PrefixTree outperforms LM algorithm, which is revised to mine moving sequences, in mining large moving sequence databases.

Keywords

Clustering, Sequential pattern, Moving sequential pattern, Data mining, Mobile Computing

* Supported by the National High Technology Development 863 Program of China under Grant No. 2002AA4Z3440; the National Grand Fundamental Research 973 Program of China under Grant No. G1999032705; the Foundation of the Innovation Research Institute of PKU-IBM.

1. Introduction

In order to provide effective and efficient location management technologies in wireless communication systems, researchers recently tend to focus on characterizing the mobile users moving and/or calling characteristics, such as CMR (Call to Mobility Ratio) [1], Markov model [2], probability distribution based on profile [3], mobility graph [4], and moving behaviors [5,6,7]. User's moving behaviors can help to allocate personal data, pre-fetch useful information, pre-allocate wireless resources, and design personal paging areas, etc. In this paper we focus on mining mobile user maximal moving sequential patterns.

Moving sequential patterns is a kind of moving behaviors, and we will first systematically describe the problem of mining moving sequential patterns. It can be viewed as a special case of mining sequential patterns with the extension of support, which helps a more reasonable pattern discovery. There are four major differences when mining conventional sequential patterns and moving sequential patterns. Firstly, if two items are consecutive in a moving sequence α , which is a subsequence of β , those two items must be consecutive in β . That is because we care about what the next move is for a mobile user in a mobile computing environment. Secondly, in mining moving sequential patterns the support considers the number of occurrences in a moving sequence, so the support of a moving sequence is the sum of the number of occurrence in all the moving sequences in the moving sequence database. Thirdly, the Apriori property plays an important role for efficient candidate pruning in mining conventional sequential patterns. For example, suppose $\langle ABC \rangle$ is a frequent length-3 sequence, and then all the length-2 subsequences $\{\langle AB \rangle, \langle AC \rangle, \langle BC \rangle\}$ must be frequent in mining sequential patterns. In mining moving sequential patterns $\langle AC \rangle$ need not be frequent. This is because a mobile user can only move into a neighboring cell in a wireless system and items must be consecutive in mining moving sequential patterns. In addition, $\langle AC \rangle$ is not a subsequence of $\langle ABC \rangle$ any more in mining moving sequential patterns, and so the property (any subsequence of a frequent moving sequence must be frequent) is still fulfilled from this meaning, which is called Pseudo-Apriori property. The last difference is that a moving sequence is an order list of items, but not an order list of itemsets, where each item is a cell id.

The problem of mining conventional sequential patterns was first introduced by R. Agrawal et al. in [9], and there have been many studies on efficient sequential pattern mining and its applications [10-14], which can be categorized into two classes: (1) Apriori-based [9-12]; (2) Projection-based [13,14]. Wen-Chih Peng et al. present a data-mining algorithm, which involves mining for user moving patterns in a mobile computing environment in [5]. Moving pattern mining is based on a roundtrip model [8], and their LM algorithm selects an initial location S , which is either VLR (Visitor Location Register) or HLR (Home Location Register) whose geography area contains the homes of the mobiles users. Suppose a mobile user goes on errands to a strange place for one month or longer, the method in [5] cannot find the proper moving pattern to characterize the mobile user. A more general method should not give the assumption of the start point of a moving pattern. Basically, algorithm LM is a variant one from GSP [9,10]. The Apriori-based methods can efficiently prune candidate sequence patterns based on Apriori property, but in moving sequential pattern mining we cannot prune candidate sequences efficiently because the moving sequential pattern only preserves Pseudo-Apriori property. In the meanwhile Apriori-based algorithms still encounter problem when a sequence database is large and/or when

sequential patterns to be mined are numerous and/or long [14]. J. Han et al. propose a method, named FP-tree to mine frequent itemsets without candidate generation in [15], and FP-tree compresses the database into a set of conditional databases (a special kind of projected database), each associated with one frequent item, and mine each database separately. Projection-based methods adopt a divide and conquer idea to confine the search and the growth of subsequence fragments, and leads to efficient processing.

Time factor is considered for personal paging area design and location management respectively in [6,7]. Time is also a very important factor in mining moving sequential patterns. G. Das et al. present a clustering based method to discretize a times series in [16], and Hsiao-Kuang Wu et al. also use clustering methods to preprocess the moving logs in personal paging area design [6]. We introduce the idea of clustering into the mining of moving sequential patterns. The idea of using clustering to discretize time is that if a mobile user possesses regular moving behavior, then he often moves on the same set of paths, and the arrival time to each point of the paths is similar. In the meanwhile, day is a natural periodic cycle that a mobile user usually behaves, so we consider the moving histories in a day. Thus the moving sequential patterns reflect a mobile user's moving behaviors in a day unit.

In this paper, firstly a clustering algorithm CURD [15] is used to discretize the time attribute of the moving histories, and the moving histories are transformed into moving sequences based on the clustering result. Then based on the idea of projection and Pseudo-Apriori property, we propose an efficient moving sequential pattern mining algorithm PrefixTree, which can effectively represent candidate frequent moving sequences with a key tree structure of prefix trees. Prefix tree is a compact representation of candidate moving sequential patterns. It is easy for us to generate the moving sequential patterns based on the prefix trees. Every moving sequence from the root node to the leaf node is a candidate frequent moving sequences. We can get all the moving sequential patterns by scanning all the prefix trees once. The support of each node decreases with the depth increase, so a new frequent moving sequence is generated when we traverse the prefix trees from the root to the leaves when encountering a node whose count is less than the support threshold. We also use a wireless network topology based optimization method to improve the efficiency of PrefixTree algorithm.

Contributions. In this paper, a novel and efficient technique is proposed to mine mobile user maximal moving sequential patterns.

Firstly, the idea of clustering is introduced into the mining of moving sequential patterns, whose main role is to discretize the time attribute as a preprocessing step.

Secondly, an efficient algorithm, called PrefixTree, is proposed to discover the mobile users' moving sequential patterns based on prefix trees. Being different from traditional projection-based sequential algorithms [13,14], PrefixTree doesn't generate projected physical files, which is time-cost work. In addition, PrefixTree only need scan the database three times, which guarantees its efficiency.

Thirdly, performance study shows that PrefixTree outperforms LM, which is revised to mine moving sequences, in mining large moving sequence databases.

Organization. The rest of the paper is organized as follows. Data preprocessing is given in section 2. In section 3, we describe PrefixTree algorithm. In section 4, we show the experimental results from different viewpoints. Discussion is made in section 5.

2. Data Preprocessing

User moving history is the moving logs of a mobile user, which is an ordered (c, t) list where c is the cell ID and t is the time when the mobile user reaches cell c . Let $MH = \langle (c_1, t_1), (c_2, t_2), (c_3, t_3), \dots, (c_n, t_n) \rangle$ be a moving history, and MH means the mobile user enters c_1 at t_1 , leaves c_1 and enters c_2 at t_2 , leaves c_2 and enters c_3 at t_3 ...and finally enters c_n at t_n . Each $(c_i, t_i) \in MH$ ($1 \leq i \leq n$) is called an element of MH .

The time difference between two consecutive elements in a moving history reflects a mobile users' moving speed (or sojourn time in a cell). If the difference is high, it shows the mobile user moves at a relative low speed; otherwise, if the difference is low, it shows the mobile user moves at a relative high speed.

ID	Moving History
1	$\langle (0, 0)(3, 363)(2, 373)(6, 393)(4, 924)(5, 987)(2, 993)(7, 999)(0, 1005) \rangle$
2	$\langle (0, 0)(3, 438)(2, 448)(7, 458)(0, 882) \rangle$
3	$\langle (0, 0)(3, 383)(2, 393)(6, 403)(4, 770)(5, 795)(2, 801)(7, 807)(0, 813) \rangle$
4	$\langle (0, 0)(8, 508)(1, 726)(9, 857)(1, 867)(9, 1163)(0, 1169) \rangle$
5	$\langle (0, 0)(8, 551)(1, 1174)(9, 1180)(0, 1186) \rangle$
6	$\langle (0, 0)(8, 412)(1, 1022)(9, 1028)(0, 1034) \rangle$

Table 1 Moving History Database

Table 1 shows us a mobile user's moving histories in six days, where the cell set is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and minute is used as the unit of time with the scope from 0 to 1439 in a day period. The following analysis will use the above moving history database as an example to explain the idea of our solution in the whole paper.

2.1 Clustering Processing

There are many clustering algorithms, such as [15, 17], and many mining tools, such as DB2 intelligent miner for data. A clustering method called CURD (Clustering Using Reference and Density) was presented in [15], which preserves the ability of density based clustering methods' good advantages, and is very efficient because of its nearly linear time complexity (see [15] for details). Here we use CURD algorithm to discretize the time attribute. The clustering processing is explained as follows,

For each cell c in the cell set, we collect all the elements in the moving history database D , and we denote the element set of c by $ES(c) = \{(c, t) | \exists (c, t) \in MH_i \wedge MH_i \in D\}$. Then we use CURD algorithm to cluster the element set $ES(c)$, where Euclidean distance on time t is used as the similarity function between two elements in $ES(c)$.

This is a clustering problem in one-dimension space. After the clustering processing we have a clustering result as $\{(c, T_s, T_e) | T_s, T_e \in T \wedge T_s \leq T_e\}$ (T is the time domain), which means the mobile user often enters cell c at the period $[T_s, T_e]$. All the clustering results of the moving history database D are shown in Table 2.

2.2 Data Transformation

The main idea of data transformation is to replace the MH elements with the corresponding clusters that they belong to.

Cluster ID	(Cell ID, T_s , T_e)
C_{01}	(0, 0, 0)
C_{02}	(0, 813, 1186)
C_1	(1, 726, 1174)
C_{21}	(2, 373, 448)
C_{22}	(2, 801, 993)
C_3	(3, 363, 438)
C_4	(4, 770, 924)
C_5	(5, 795, 987)
C_6	(6, 393, 403)
C_7	(7, 807, 999)
C_8	(8, 412, 551)
C_9	(4, 857, 1180)

Table 2 Clustering Results

ID	Moving Sequence
1	$\langle C_{01}C_3C_{21}C_6C_4C_5C_{22}C_7C_{02} \rangle$
2	$\langle C_{01}C_3C_{21}C_7C_{02} \rangle$
3	$\langle C_{01}C_3C_{21}C_6C_4C_5C_{22}C_7C_{02} \rangle$
4	$\langle C_{01}C_8C_1C_9C_1C_9C_{02} \rangle$
5	$\langle C_{01}C_8C_1C_9C_{02} \rangle$
6	$\langle C_{01}C_8C_1C_9C_{02} \rangle$

Table 3 Moving Sequence Database

The transformed moving history is called a moving sequence, and the transformed moving history database is called moving sequence database. After transformation the moving history database in Table 2 becomes the moving sequence database in Table 3.

Lemma 1: Each moving history can be transformed into one and only one moving sequence.

Proof. The clustering method guarantees that any MH element belongs to one and only one cluster, so the above lemma is true.

3. Mining Moving Sequential Patterns

3.1 Related Definitions

Let $C = \{CT_1, CT_2, \dots, CT_n\}$ be a set of all items, each CT_i is a tuple, denoted by (c, T_s, T_e) , where c is a cell id, T_s and T_e are time points, and $T_s \leq T_e$. A moving sequence is an order list of items, and is denoted by $\langle s_1 s_2 \dots s_m \rangle$ ($s_i \in C$), and each moving sequence records the moving behavior of a mobile user with the same period, such as a day, a month, etc. The number of items in a sequence is called the length of the sequence. A moving sequence with length k is called a k -sequence. A moving sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$ is called a subsequence of another moving sequence $\beta = \langle b_1 b_2 \dots b_m \rangle$ and β is a super sequence of α , denoted by $\alpha \subseteq \beta$, if there exists an integer $1 \leq i \leq m$ such that $a_1 = b_i, a_2 = b_{i+1}, \dots, a_n = b_{i+n-1}$.

A moving sequence database D is a set of tuples $\langle \text{sid}, s \rangle$, where sid is the sequence id and s is a moving sequence. A tuple $\langle \text{sid}, s \rangle$ is said to contain a sequence α if $\alpha \subseteq s$, and the number of occurrences of α within s is denoted by $C_{\langle \text{sid}, s \rangle}(\alpha)$. The support of a sequence α in a moving sequence database D is denoted by $\text{support}(\alpha) = \sum(C_{\langle \text{sid}, s \rangle}(\alpha))$, where $\langle \text{sid}, s \rangle \in D$ and $\alpha \subseteq s$. Give a positive integer ζ as the support threshold, a moving sequence α is called a moving sequential pattern, if the support of α is at least ζ in the database, i.e. $\text{support}(\alpha) \geq \zeta$. A moving sequential pattern α is called a maximal moving sequential pattern if there is not any other moving sequential pattern β , and $\alpha \subseteq \beta \wedge \alpha \neq \beta$.

Lemma 2: A moving sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$ of length n has $n-k+1$ subsequences of length k ($1 \leq k \leq n$).

Proof. From the definition of subsequence it is easy to know that the above lemma is true.

Lemma 3: A moving sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$ of length n has 2 subsequences of length $n-1$.

Proof. It follows from lemma 2, if $k=n-1$, and it is easy to know that the above lemma is true.

The above two lemmas give the theoretic evidence that the Apriori-based methods fail pruning candidate sequence patterns efficiently in mining moving sequential patterns. Any length n ($n > 1$) candidate moving sequence must be generated from two frequent length $n-1$ moving sequences. As Lemma 2 shows length n sequence only has 2 subsequence of length $n-1$ and those subsequences must be frequent, so no candidate sequences can be pruned at all. The above lemmas also give the reason why Pseudo-Apriori property is preserved in mining moving sequential patterns.

Definition 1 (Prefix, projection, and postfix)

Given a moving sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$, a moving sequence $\beta = \langle b_1 b_2 \dots b_m \rangle$ ($m \leq n$) is called a **prefix** of α if and only if $b_i = a_i$ for $(1 \leq i \leq m)$.

Given moving sequences α and β such that β is a subsequence of α , i.e., $\beta \subseteq \alpha$. A subsequence α' of α (i.e. $\alpha' \subseteq \alpha$) is called a **projection** of α w.r.t. prefix β if and only if (1) α' has prefix β and (2) there is no proper super-sequence α'' of α' (i.e. $\alpha' \subseteq \alpha''$ but $\alpha' \neq \alpha''$) such that α'' is a subsequence of α and also has prefix β .

Let $\alpha' = \langle c_1 c_2 \dots c_l \rangle$ be the projection of α w.r.t. prefix $\beta = \langle c_1 c_2 \dots c_k \rangle$ ($k \leq l$). Sequence $\gamma = \langle c_{k+1} \dots c_l \rangle$ is called the **postfix** of α w.r.t. prefix β , denoted by $\gamma = \alpha / \beta$. If $\alpha' = \beta$, postfix of α w.r.t. β are empty, and if β is not a subsequence of α , both projection and postfix of α w.r.t. β are empty.

From the above definitions, we can see they are similar to the ones in PrefixSpan [14], and are changed according to the characteristics of moving sequences.

Definition 2 (Before relation and After relation)

For any two items, their time attributes have the following two relations:

For two items $C_i = (c_k, T_{si}, T_{ei})$ and $C_j = (c_m, T_{sj}, T_{ej})$, if $(T_{ei} < T_{sj})$, we call item C_i is **before** item C_j , denoted by $C_i \vdash C_j$.

For two items $C_i = (c_k, T_{si}, T_{ei})$ and $C_j = (c_m, T_{sj}, T_{ej})$, if $(T_{ei} \geq T_{sj})$, we call item C_i is **after** item C_j , denoted by $C_i \dashv C_j$.

From the above definitions, we can see that for any two items must fulfill either before or after relation, and if they fulfill before relation they cannot fulfill after relation, vice versa. We now give an intuitionistic explanation about the before and after relations, suppose two items C_i and C_j both appear in a moving sequence. If $C_i \vdash C_j$, C_i must appear before C_j in any other moving sequence, and if $C_i \dashv C_j$, C_i may appear after C_j in other moving sequences (sometimes C_i may appear before C_j).

Definition 3 (Candidate Consecutive Items)

For each item C_i , we call the items that may appears just after it candidate consecutive items. It is easy to know that only the items after C_i in a moving sequence may be the consecutive items of C_i , denoted by $CCI(C_i)$, and $CCI(C_i) = \{C_j | C_j \dashv C_i\}$.

3.2 PrefixTree Algorithm

The mining of mobile user maximal moving sequential patterns is divided into six steps as follows,

Step 1. Scanning the database once to find the set of frequent items, and then order them according to their values of T_s attribute.

Step 2. Computing candidate consecutive items in terms of the after relation.

Step 3. Scanning the database again to generate frequent length-2 moving sequences, where the candidate frequent length-2 moving sequences are generated based on candidate consecutive items. Based on the frequent length-2 moving sequences, candidate consecutive items are reduced again based on Pseudo-Apriori property.

Step 4. Scanning the database to construct prefix trees based on candidate consecutive items and Pseudo-Apriori property, which is the last time to scan the database.

Step 5. Generating moving sequential patterns based on the prefix trees.

Step 6. Generating maximal moving sequential patterns.

We will use the example of moving sequence database D in Table 3 with support=3 to show the details of the above six steps one by one.

Step 1 (Generating Frequent Items)

This processing is similar to normal sequential pattern mining. PrefixTree first scans D, collects the support for each item, and find the frequent items. Frequent items are listed in an ascending order according to their T_s attribute in the form of item: support as below,

$$F_ItemList = \{C_{01}: 6, C_3: 3, C_{21}: 3, C_8: 3, C_1: 4, C_7: 3, C_{02}: 6, C_9: 4\}.$$

Those frequent items are also the frequent length-1 moving sequences.

Step 2 (Computing Candidate Consecutive Items)

Each item has two time attributes T_s and T_e , which form a time region that reflects a mobile user often enters some cell area at some time between T_s and T_e . We compute the frequent items' candidate consecutive items according to the after relation. This processing is very simple. First, we order the frequent items in an ascending order according to their T_e attribute. Second, for each frequent item, the frequent items, who's T_e are equal to or bigger than it's T_s , are listed in its CCI. The details of the frequent items' CCIs are showed in Table 4. This step helps to decrease the number of candidate length-2 moving sequential patterns in the next step, and then improves the processing efficiency.

ID	CCIs
C_{01}	$C_{01}C_3C_{21}C_8C_7C_1C_9C_{02}$
C_3	$C_3C_{21}C_8C_7C_1C_9C_{02}$
C_{21}	$C_3C_{21}C_8C_7C_1C_9C_{02}$
C_8	$C_3C_{21}C_8C_7C_1C_9C_{02}$
C_1	$C_7C_1C_9C_{02}$
C_7	$C_7C_1C_9C_{02}$
C_{02}	$C_7C_1C_9C_{02}$
C_9	$C_7C_1C_9C_{02}$

Table 4 Initial CCIs

Step 3 (Generating Length-2 Moving Sequential Patterns)

To any item it may have the same number of candidate length-2 moving sequential patterns as the number of all the frequent items. The candidate consecutive items are used to limit the number of candidate length-2 moving sequences, which improves the processing efficiency. For example, the candidate length-2 moving sequential patterns with prefix $\langle C_{01} \rangle$ are $\{\langle C_{01}C_i \rangle | C_i \in CCI(C_{01})\}$. By scanning once the moving sequence database D , we can compute the support for all candidate length-2 moving sequential patterns. Now we have the complete set of length-2 moving sequential patterns in the form of sequence: support as bellow,

$$F_Length2List = \{\langle C_{01}C_3 \rangle: 3, \langle C_{01}C_8 \rangle: 3, \langle C_3C_{21} \rangle: 3, \langle C_8C_1 \rangle: 3, \langle C_1C_9 \rangle: 4, \langle C_7C_{02} \rangle: 3, \langle C_9C_{02} \rangle: 3\}.$$

After getting all the length-2 moving sequential patterns, The CCI of each frequent item can be optimized again. Suppose there is a length- n moving sequential pattern $\alpha = \langle a_1a_2 \dots a_n \rangle$, and we will generate all the length- $(n+1)$ moving sequential patterns with prefix α . All the candidate frequent moving sequences have the form of $\langle a_1a_2 \dots a_nb \rangle$, and if $\langle a_1a_2 \dots a_nb \rangle$ is frequent, $\langle a_nb \rangle$ must be frequent based on the Pseudo-Apriori property. According to the definition of CCI it is easy to know that $b \in CCI(a_n)$, so we get the rule that for any item $b \in CCI(a)$, $\langle ba \rangle$ must be frequent. Thus candidate consecutive items of each item can be reduced again, which is shown in Table 5. We can see that this results in a huge reduction of the CCI's scale. The behind fact is that a mobile user can only move into the current cell's neighbor cells. The number of neighbor cells is often small, for example it is two in one-dimension model, and six in two-dimension hexagonal model, and eight in two-dimension mesh model, and is relative small even in graph model [20]. Any frequent moving sequence α with C as its last item cannot grow into longer frequent moving sequences having prefix α if the CCI of C is empty.

ID	CCIs
C_{01}	C_3C_8
C_3	C_{21}
C_{21}	\emptyset
C_8	C_1
C_1	C_9
C_7	C_{02}
C_{02}	\emptyset
C_9	C_{02}

Table 5 Reduced CCIs

Step 4 (Constructing Prefix Trees)

In this step we will construct a prefix tree for each frequent item. The moving sequential patterns, i.e. the complete set of frequent moving sequences, can be divided into the following eight subsets according to the eight prefixes: (1) the one having prefix $\langle C_{01} \rangle$; ...and (8) the one having prefix $\langle C_9 \rangle$. The eight prefixes are ordered by their value of attribute T_s .

Prefix tree is a compact representation of candidate moving sequential patterns, which has the following characteristics:

- 1) The root is the frequent item, and is defined at depth one;

- 2) Each node contains three attributes: one is the item, one is the count attribute which means the support of the item, and the last one is the flag indicating whether the node is traversed;
- 3) The items of a node's children are all contained in its candidate consecutive items.

Firstly, the prefix trees are initialized based on the frequent items and the frequent length-2 moving sequences. Secondly, by scanning the database once, all the prefix trees, which can effectively represent the complete set of candidate frequent moving sequences, can be constructed. The algorithms of constructing prefix trees are shown as follows,

Algorithm1 GPS (Generating Projected Sequences)

Input: Moving sequences α and β

Output: The projected sequences of α w.r.t. β .

1. $S = \emptyset$;
 2. $\alpha' = \text{postfix}(\alpha, \beta)$; //postfix (α, β) is a function, which returns the postfix of α w.r.t. β
 3. While (α' is not empty) {
 4. $S = S \cup \{\alpha'\}$;
 5. $\alpha' = \text{postfix}(\alpha', \beta)$;
 6. }
 7. Return S
-

Algorithm2 ISPT (Insert a Sequence into a Prefix Tree)

Input: A moving sequence α , and a prefix tree PT

Output: a prefix tree PT'

1. If the root node N_R of PT contains no children Then break;
 2. For every child node N_C of N_R {
 3. $\beta = \langle N_R.\text{item} N_C.\text{item} \rangle$;
 4. $S = \text{GPS}(\alpha, \beta)$;
 5. For every sequence α' in S {
 6. b is item that is contained in the current node;
 7. While (α is not empty) {
 8. c is the first item of α' ;
 9. If some child node of the current one that contains c, just increase the node's support by one;
 10. Else If $c \in \text{CCI}(b)$, generate a new node as the child node of the current one;
 11. Otherwise, stop the while loop;
 12. $\alpha' = \text{postfix}(\alpha', \langle c \rangle)$;
 13. b: = c;
 14. }
 15. }
 16. }
 17. Return PT
-

Algorithm3 CPT (Constructing Prefix Trees)

Input: A moving sequence database D, the frequent items set FSet, and the set of CCIs

Output: The complete set of prefix trees

1. Initializing prefix trees PTSet according to FSet and CCIs;
 2. For every moving sequence α in D {
 3. For every initial prefix tree pt in PTSet
 4. **ISPT** (α, pt);
 5. }
 6. Return PTSet
-

Given moving sequences α , algorithm CPT is used to add the information contained in α to all the existing prefix trees; Algorithm ISPT is really used to add the information contained in α to a prefix tree; And algorithm GPS is used to generate the projected sequences. All this work can be done in one scan of the moving database, which guarantees its efficiency. The novelty of PrefixTree is that it doesn't generate projected physical files, which is different from traditional projection-based sequential algorithms [13,14] and is a time-cost work.

Step 5 (Generating Moving Sequential Patterns)

Based on the prefix trees, it is easy for us to generate the moving sequential patterns. Every moving sequence from the root node to the leaf node is a candidate frequent moving sequences. Scanning all the prefix trees once can get all the moving sequential patterns. Here the depth first searching strategy is used to find all the frequent moving sequences in a prefix tree. The support of each node decreases with the depth increase, so a new frequent moving sequence is generated when traversing the prefix trees from the root to the leaves when encountering a node whose count is less than the support threshold.

Step 6 (Generating Maximal Moving Sequential Patterns)

This step is to delete the moving sequential patterns that are contained in other moving sequential patterns. The method of maximal phase in [9] can be used to get the final maximal moving sequential patterns.

3.3 Optimization of PrefixTree Algorithm

As pointed in [21], the initial candidate set generation, especially for the length-2 frequent patterns, is the key issue to improve the performance of data mining. Fortunately, sometimes the apriori of the wireless network topology can be got. From the apriori we can know the neighboring cells for each cell. We have pointed out that for any two consecutive items in a moving sequence, one item must be the other one's neighboring cell or itself. The number of neighbor cells is often small, for example it is two in one-dimension model, and six in two-dimension hexagonal model, and eight in two-dimension mesh model, and is relative small even in graph model [20]. So based on this apriori the number of the candidate length-2 moving sequential patterns is decreased much, and then we can efficiently generating the frequent length-2 moving sequential patterns in step 3. In fact, this optimization method can be used for other algorithms too, and we show how efficient it is in later experiments.

3.4 Analysis of PrefixTree Algorithm

Based on Pseudo-Apriori property, for any frequent length- n moving sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$, all the possible frequent length- $(n+1)$ moving sequences having prefix α have the form of $\langle a_1 a_2 \dots a_n b \rangle$, and if $\langle a_1 a_2 \dots a_n b \rangle$ is frequent, $\langle a_n b \rangle$ must be frequent based on the Pseudo-Apriori property, so b must belong to $CCI(a_n)$. From the above description and analysis of PrefixTree algorithm, it is easy to know the prefix trees contain all the possible frequent moving sequences, and PrefixTree can discover the complete set of maximal moving sequential patterns.

PrefixTree needs to scan the moving sequence database once to generate the frequent items and scan the database again to generate the frequent length-2 moving sequences. Then PrefixTree constructs the prefix trees by scanning the database for the last time, thus in all PrefixTree only needs scan the database three times, which guarantees the efficiency of PrefixTree algorithm.

4. Experimental Results and Performance Study

All experiments are performed on a 1.7GHz Pentium 4 PC machine with 512M main memory and 60G hard disk, running Microsoft Windows 2000 Professional. All the methods are implemented using JBuilder 6.0.

The synthetic dataset used in our experiments comes from SUMATRA (Stanford University Mobile Activity TRAcEs) [22]. BALI-2: Bay Area Location Information (real-time) dataset records the mobile users' moving and calling activities in a day. The mobile user averagely moves 7.2 times in a day in 90 zones, so the number of items is 90 and the average length of the moving sequences is 8.2. BALI-2 contains about 40,000 moving sequences, which are used for our

experiments.

Since the details of experimental analysis of CURD are shown in [15], here we focus on the experimental analysis of PrefixTree. We report our experimental results on the performance of PrefixTree in comparison with our version of LM, called Revised LM. Revised LM first computes the frequent items and length-2 moving sequential patterns, and then it uses the original LM to find the moving sequential patterns, and lastly the maximal moving sequential patterns are found. The Revised LM uses the same method as PrefixTree to find the frequent items and length-2 moving sequential patterns, and uses the same method to find the maximal moving sequential patterns (see [5] for details about LM).

Before showing the experimental results, we first give a simple I/O analysis of PrefixTree and Revised LM. Let $|D|$ is the number of moving sequences in database D , and L is the length of the longest moving sequential pattern. Let reading a moving sequence in a data file costs 1 unit of I/O. The I/O cost of Revised LM is equal to $L(|D|)$; the I/O cost of PrefixTree is equal to $3(|D|)$. From the I/O costs analysis we could get a coarse conclusion that PrefixTree is more efficient than LM if L is bigger than 3.

We have stated that the average length of the moving sequences is 8.2, so the length is a relative small number. Even so, PrefixTree still outperforms Revised LM in our experiments.

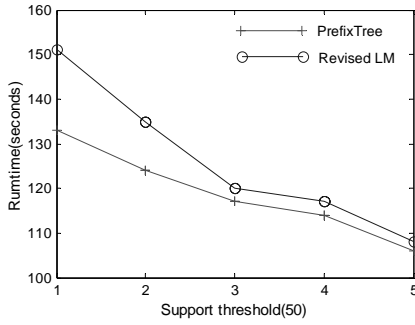


Figure 8 PrefixTree and Revised LM

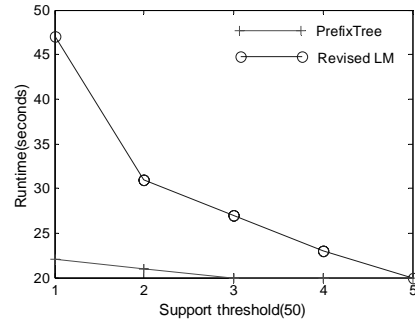


Figure 9 PrefixTree and Revised LM with constraint

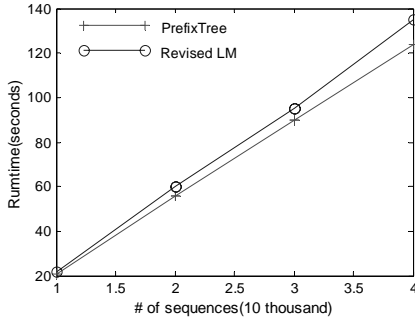


Figure 10 PrefixTree and Revised LM

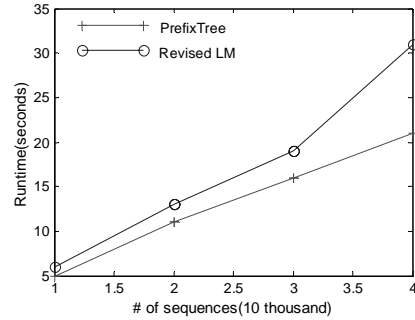


Figure 11 PrefixTree and Revised LM with constraint

The experimental results of scalability of PrefixTree and Revised LM according to the support threshold are shown in Figure 8 and Figure 9, and the number of moving sequences is about 40,000. Figure 8 and Figure 9 show the experimental results without and with the wireless network topology constraint respectively. When the support is high, there is only a limited number of moving sequential patterns, and the length of patterns is short, the two methods are close to each other according to their runtime. However as the support threshold decreases, PrefixTree is more efficient than Revised LM because a lower support threshold results in longer moving sequential patterns. If we compare Figure 8 with Figure 9, we can find that the run time of PrefixTree with constraint is about 1/6 of PrefixTree and the one of Revised LM with constraint is

about 1/3 of Revised LM. That proves the efficiency of the wireless network topology based optimization method.

Figure 10 and Figure 11 show the experimental results of the scalability of PrefixTree and Revised LM in terms of the number of moving sequences, and the support threshold is set to 100. Figure 10 and Figure 11 show the experimental results without and with the wireless network topology constraint respectively. Both methods are linearly scalable, but PrefixTree is more efficient than Revised LM. If we compare Figure 10 with Figure 11, we can find that the run time of PrefixTree with constraint is about 1/5 of PrefixTree and the one of Revised LM with constraint is about 1/4 of Revised LM. That proves the efficiency of the wireless network topology based optimization method again.

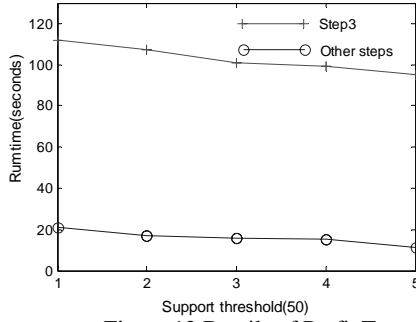


Figure 12 Details of PrefixTree

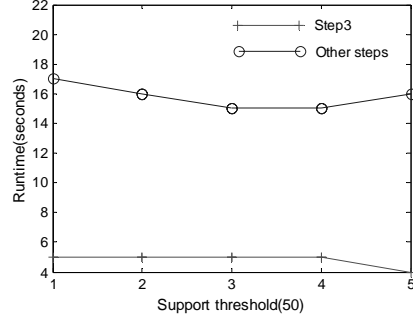


Figure 13 Details of PrefixTree with constraint

Figure 12 and Figure 13 show the executing details of PrefixTree from the aspect of support threshold, and the number of moving sequences is about 40,000. Figure 12 shows the result without the wireless network topology constraint, and Figure 12 shows that the step 3, which needs to generate frequent length-2 moving sequences, occupies the main cost of PrefixTree. Compared with step 3 the other steps of PrefixTree occupy only a little time. Figure 13 shows the experimental result with the wireless network topology constraint and Figure 13 shows that the step 3 does not occupy the main cost of PrefixTree any more. Compared with the run time of step 3 in Figure 12, the one of step 3 in Figure 13 is much small. The run time of step 3 with the wireless network topology constraint is only about 1/20 of the one of step without optimization, which shows the efficiency and effectivity of the optimization method.

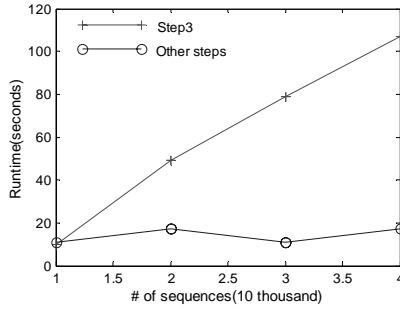


Figure 14 Details of PrefixTree

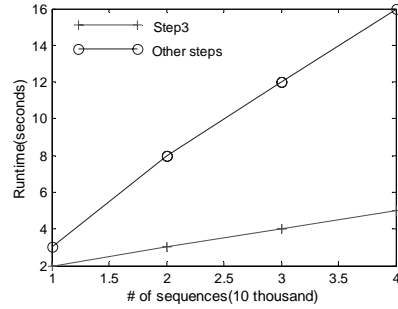


Figure 15 Details of PrefixTree with constraint

Figure 14 and Figure 15 show the executing details of PrefixTree from the aspect of the number of moving sequences, and the support threshold is set to 100. Figure 14 shows the result without the wireless network topology constraint, and Figure 14 shows that the step 3, which needs to generate frequent length-2 moving sequences, occupies the main cost of PrefixTree. Compared with step 3 the other steps of PrefixTree occupy only a little time. Figure 15 shows the result with the wireless network topology constraint and Figure 15 shows that the step 3 does not occupy the main cost of PrefixTree. Compared with the run time of step 3 in Figure 14, the one of

step 3 in Figure 15 is much small. The run time of step 3 with the wireless network topology constraint is only about 1/20 of the one of step without optimization, which shows the efficiency and effectivity of the optimization method again.

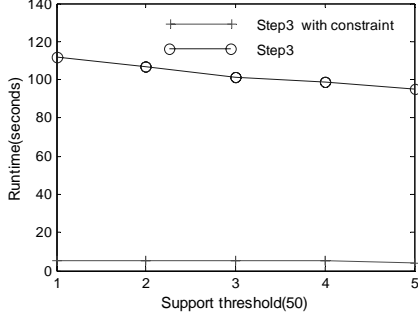


Figure 16 Comparison of Step 3

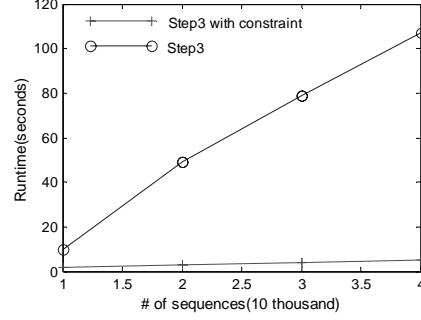


Figure 17 Comparison of Step 3

Figure 16 and Figure 17 show the experimental results of the comparison of step 3 of PrefixTree from the aspects of with or without the wireless network topology constraint. The run time of step 3 with the wireless network topology constraint is only about 1/20 of the one of step without optimization. Both the experimental results show that the optimization based on the wireless network topology improves much the efficiency of generating the length-2 moving sequential patterns and the whole algorithm.

In summary, performance study shows that PrefixTree algorithm, which needs only three scans of the moving sequence database, is more efficient and scalable than Revised LM, which needs multiple scans of the moving sequence database limited with the length of the longest moving sequential pattern. In addition, the optimization based on the wireless network topology improves much the efficiency of mining processing.

5. Discussion

In this paper a clustering method is introduced to discretize the time attribute in moving histories, which transforms the original moving histories into moving sequences based on the clustering results. And then a novel and efficient method, called PrefixTree, is proposed to mine the moving sequences. Its main idea is to generate projected sequences and construct the prefix trees. Performance study shows that PrefixTree is more efficient and scalable than Revised LM.

Another valuable function of PrefixTree is supporting support threshold tuning. It is highly desirable because in most cases the user tends to try a few minimum supports before being satisfied with the result. Pruning the prefix trees with a smaller support threshold, which avoid re-mining the whole moving sequence database, can generate the prefix trees with higher support threshold.

Acknowledgement

We are very grateful to Professor Lu Hong Jun for giving us many good suggestions and advices, which help improve the quality of this paper; we are also thankful to Professor Pei Jian for helping us to understand the PrefixSpan algorithm.

References

- [1] T.Imielinski, B.R.Badrinath. Querying in Highly Mobile Distributed Environments. Proc. of VLDB Conference, pp. 41-52, 1992.
- [2] A. Bar-Noy, I. Kessler, and M. Sidi. Mobile Users: To Update or not to Update? ACM/Baltzer J. Wireless Networks, vol. 1, no. 2, pp. 175-195, 1995.
- [3] Sami Tabbane. An Alternative Strategy for Location Tracking. IEEE Journal on Selected Areas in Comms, vol. 13, pp. 880 - 892, 1995.
- [4] Zohar Naor, Hanoch Levy. Minimizing the Wireless Cost of Tracking Mobile Users: An Adaptive Threshold Scheme. Proc. of INFOCOM Conference, pp. 720-727, 1998.
- [5] Wen-Chih Peng, Ming-Syan Chen. Mining User Moving Patterns for Personal Data Allocation in a Mobile Computing System. Proc. of ICPP Conference, pp. 573-580, 2000.
- [6] Hsiao-Kuang Wu, Ming-Hui Jin, Jorng-Tzong Horng. Personal Paging Area Design Based On Mobiles Moving Behaviors. Proc. of INFOCOM Conference, pp. 21-30, 2001.
- [7] Wenchao Ma and Yuguang Fang. A New Location Management Strategy Based on Mobility Pattern for Wireless Networks. Proc. of LCN Conference, pp.451-457, 2002.
- [8] N. Shivakumar, J. Jannink, and J. Widom. Per-user Profile Replication in Mobile Environments: Algorithms Analysis and Simulation Result. ACM/Baltzer Journal of Mobile Networks and Applications, 2(2): 129-140, 1997.
- [9] Rakesh Agrawal, Ramakrishnan Srikant. Mining Sequential Patterns. Proc. of ICDE Conference, pp.3-14, 1995.
- [10] Ramakrishnan Srikant, Rakesh Agrawal, Mining Sequential patterns: Generalizations and Performance Improvements. Proc. of EDBT Conference, pp.3-17, 1996.
- [11] Mohammed J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning, 0, 1-31 (2000) Kluwer Academic Publishers.
- [12] Jay Ayres, Johannes Gehrke, Tomi Yu, and Jason Flannick. Sequential Pattern Mining using A Bitmap Representation. Proc. of KDD Conference, pp. 429-435, 2002.
- [13] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl et al. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. Proc. of KDD Conference, pp.355-359, 2000.
- [14] J. Pei, J. Han, B. Mortazavi-Asl et al. PrefixSpan: Mining Sequential Patterns Efficiently by PrefixProjected Pattern Growth. Proc. of ICDE Conference, pp. 215-224, 2001.
- [15] Shuai Ma, TengJiao Wang, ShiWei Tang et al. A New Fast Clustering Algorithm Based on Reference and Density. G Dong et al. (Eds.), Proc. of WAIM Conference, LNCS 2762, Springer-Verlag, pp.214-225, 2003.
- [16] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule Discovery from Time Series. Proc. of KDD Conference, pp. 16-22, 1998.
- [17] M. Ester, H. P. Kriegel, J. Sander et al. A Density Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proc. of KDD Conference, pp. 226-231, 1996.
- [18] Jian Pei, Jiawei Han, Wei Wang. Mining Sequential Patterns with Constraints in Large Databases. Proc. of CIKM Conference, pp. 18-25, 2002.
- [19] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. Proc. of SIGMOD Conference, pp. 1-12, 2000.
- [20] Vincent W.-S. Wong, Victor C. M. Leung. Location Management for Next Generation Personal Communication Networks. IEEE network, vol. 14, no. 5, pp. 18-24, 2000.
- [21] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. IEEE TKDE, vol.9, no.5, pp. 813-825, 1997.
- [22] SUMATRA: Stanford University Mobile Activity TRAcEs. <http://www-db.stanford.edu/sumatra/>.